Random Number Generation and Stream Cipher

GOUTAM PAUL

Asst. Professor
Department of Computer Science & Engineering
Jadavpur University, Kolkata.

July 16, 2011

Tutorial Workshop on Cryptology
(Jointly organized by: CU & Centre of Excellence in Cryptology, ISI)
Rajabazar Science College Campus, University of Calcutta, India.
Outline

1. Randomness
   - Defining Randomness
   - Testing Randomness
   - Cryptographic Randomness

2. Random Number Generation
   - Natural Random Number Generators
   - Pseudo-Random Number Generators

3. Stream Ciphers
   - Hardware Stream Ciphers
   - Software Stream Ciphers
   - Distinguisher
Roadmap

1. **Randomness**
   - Defining Randomness
   - Testing Randomness
   - Cryptographic Randomness

2. **Random Number Generation**
   - Natural Random Number Generators
   - Pseudo-Random Number Generators

3. **Stream Ciphers**
   - Hardware Stream Ciphers
   - Software Stream Ciphers
   - Distinguisher
Notion of Randomness

A numeric sequence is said to be statistically random when it contains no recognizable patterns or regularities.

Examples:
- Sequence of Head and Tail in an unbiased coin toss.
- Results of an ideal die roll.
- Digits of $\pi$.
A numeric sequence is said to be statistically random when it contains no recognizable patterns or regularities.
A numeric sequence is said to be statistically random when it contains no recognizable patterns or regularities.

Examples:
Notion of Randomness

- A numeric sequence is said to be statistically random when it contains no recognizable patterns or regularities.

Examples:
- Sequence of Head and Tail in an unbiased coin toss.
A numeric sequence is said to be statistically random when it contains no recognizable patterns or regularities.

Examples:
- Sequence of Head and Tail in an unbiased coin toss.
- Results of an ideal die roll.
A numeric sequence is said to be statistically random when it contains no recognizable patterns or regularities.

Examples:
- Sequence of Head and Tail in an unbiased coin toss.
- Results of an ideal die roll.
- Digits of $\pi$. 
It is not possible to mathematically prove that a sequence is random. It is possible to test whether a sequence is non-random.
Test of (Non-)Randomness

- It is not possible to mathematically prove that a sequence is random.
Test of (Non-)Randomness

- It is not possible to mathematically prove that a sequence is random.
- It is possible to test whether a sequence is non-random.
<table>
<thead>
<tr>
<th>Frequency Test</th>
</tr>
</thead>
</table>

**Frequency Test**

- **Checking that each symbol occurs with equal frequency.**
- For a binary string, the proportion of 0's and 1's should be 0.5 each.
- Can be generalized to $n$-gram frequencies.
Frequency Test

- Checking that each symbol occurs with equal frequency.
Frequency Test

- Checking that each symbol occurs with equal frequency.
- For a binary string, proportion of 0’s and 1’s should be 0.5 each.
Frequency Test

- Checking that each symbol occurs with equal frequency.
- For a binary string, proportion of 0’s and 1’s should be 0.5 each.
- Can be generalized to $n$-gram frequencies.
Gap Test

Look at the distances between a particular symbol. For example, for the symbol 0, 00 would be a distance of 0, 030 would be a distance of 1, 02250 would be a distance of 3, etc.
Gap Test

- Look at the distances between a particular symbol.
Gap Test

- Look at the distances between a particular symbol.
- For example, for the symbol 0,
Gap Test

- Look at the distances between a particular symbol.
- For example, for the symbol 0,
  - 00 would be a distance of 0.
  - 030 would be a distance of 1.
  - 02250 would be a distance of 3, etc.
Run Test

A run is a sequence of consecutive digits. This test is based on the frequency of run-lengths. Example: 522238 has a run of 2's of length 3.
Run Test

- A run is a sequence of consecutive digits.
Run Test

- A run is a sequence of consecutive digits.
- This test is based on the frequency of run-lengths.
Run Test

- A run is a sequence of consecutive digits.
- This test is based on the frequency of run-lengths.
- Example: 522238 has a run of 2’s of length 3.
<table>
<thead>
<tr>
<th>Randomness</th>
<th>Defining Randomness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Number Generation</td>
<td>Testing Randomness</td>
</tr>
<tr>
<td>Stream Ciphers</td>
<td>Cryptographic Randomness</td>
</tr>
</tbody>
</table>

**Autocorrelation Test**

Correlation between two sequences/processes gives a measure of similarity between them. Autocorrelation is the correlation between the measurements of the same process at two different instances of time. If random, such autocorrelations should be near zero for any and all time-lag separations.
Autocorrelation Test

- Correlation between two sequences/processes gives a measure of similarity between them.
Autocorrelation Test

- Correlation between two sequences/processes gives a measure of similarity between them.
- Autocorrelation: correlation between the measurements of the same process at two different instances of time.
Autocorrelation Test

- Correlation between two sequences/processes gives a measure of similarity between them.
- Autocorrelation: correlation between the measurements of the same process at two different instances of time.
- If random, such autocorrelations should be near zero for any and all time-lag separations.
Maurer’s Universal Test

Source modeled as
Maurer’s Universal Test

Source modeled as

- an ergodic stationary process
Maurer’s Universal Test

Source modeled as
- an ergodic stationary process
- with finite memory
Maurer’s Universal Test

Source modeled as

- an ergodic stationary process
- with finite memory
- having arbitrary (unknown) state transition probabilities.
Example with a Binary String

Consider the string 0010110011101.
Example with a Binary String

Consider the string 0010110011101.

- Frequency test:
  \[ \text{freq}(0) = 6, \text{freq}(1) = 7, \]
  \[ \text{freq}(00) = 2, \text{freq}(01) = 4, \text{freq}(10) = 3, \text{freq}(11) = 3. \]
Example with a Binary String

Consider the string 0010110011101.

- Frequency test:
  \[ \text{freq}(0)=6, \text{freq}(1)=7, \]
  \[ \text{freq}(00)=2, \text{freq}(01)=4, \text{freq}(10)=3, \text{freq}(11)=3. \]

- Gap test: \[ \text{freq}(\text{gap } 0)=2, \text{freq}(\text{gap } 1)=1, \text{freq}(\text{gap } 2)=1, \text{freq}(\text{gap } 3)=1. \]
Example with a Binary String

Consider the string 0010110011101.

- Frequency test:
  \[\text{freq}(0) = 6, \text{freq}(1) = 7,\]
  \[\text{freq}(00) = 2, \text{freq}(01) = 4, \text{freq}(10) = 3, \text{freq}(11) = 3.\]

- Gap test:
  \[\text{freq}(\text{gap 0}) = 2, \text{freq}(\text{gap 1}) = 1, \text{freq}(\text{gap 2}) = 1, \text{freq}(\text{gap 3}) = 1.\]

- Run test:
  \[\text{freq}(\text{len 1}) = 4, \text{freq}(\text{len 2}) = 3, \text{freq}(\text{len 3}) = 1.\]
Example with a Binary String

Consider the string 0010110011101.

- Frequency test:
  \[ \text{freq}(0)=6, \text{freq}(1)=7, \]
  \[ \text{freq}(00) = 2, \text{freq}(01) = 4, \text{freq}(10)=3, \text{freq}(11) = 3. \]

- Gap test: \[ \text{freq}(\text{gap 0})=2, \text{freq}(\text{gap 1})=1, \text{freq}(\text{gap 2})=1, \text{freq}(\text{gap 3}) = 1. \]

- Run test: \[ \text{freq}(\text{len 1})=4, \text{freq}(\text{len 2})=3, \text{freq}(\text{len 3})=1. \]

- Autocorrelation test:
  \[ \text{Lag 1 autocorrelation} = 3, \]
  \[ \text{Lag 2 autocorrelation} = 3. \]
Encryption increases Randomness
The goal of encryption is to make the transmitted message look random.
Perfect Secrecy
Perfect Secrecy

Information Theoretic Security:
Perfect Secrecy

Information Theoretic Security:

\[ \text{Prob}(P \mid C) = \text{Prob}(P). \]
Randomness
Random Number Generation
Stream Ciphers

Defining Randomness
Testing Randomness
Cryptographic Randomness

From Non-Random to Random-Looking


Encryption:
\[ C_i = M_i \oplus K_i. \]

Decryption:
\[ M_i = C_i \oplus K_i. \]
From Non-Random to Random-Looking

- Result: \( \text{XOR(Arbitrary bitstring, Random bitstring)} = \text{Random bitstring} \).
- Encryption \( C_i = M_i \oplus K_i \).
Result: $\text{XOR(Arbitrary bitstring, Random bitstring)} = \text{Random bitstring.}$

Encryption $C_i = M_i \oplus K_i.$

Decryption: $M_i = C_i \oplus K_i.$
One Time Pad

A different keystream is XOR-ed with each different plaintext message.

Has the property of perfect secrecy.
One Time Pad

- A different keystream is XOR-ed with each different plaintext message.
One Time Pad

- A different keystream is XOR-ed with each different plaintext message.
- Has the property of perfect secrecy.
One Time Pad

- A different keystream is XOR-ed with each different plaintext message.
- Has the property of perfect secrecy.
Roadmap

1. Randomness
   - Defining Randomness
   - Testing Randomness
   - Cryptographic Randomness

2. Random Number Generation
   - Natural Random Number Generators
   - Pseudo-Random Number Generators

3. Stream Ciphers
   - Hardware Stream Ciphers
   - Software Stream Ciphers
   - Distinguisher
# Necessity

One Time Pad requires a long stream of random bits. Other cryptographic schemes also require random numbers as keys.
Necessity

- One Time Pad requires a long stream of random bits.
One Time Pad requires a long stream of random bits.
Other cryptographic schemes also require random numbers as keys.
<table>
<thead>
<tr>
<th>Randomness</th>
<th>Natural Random Number Generators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Number Generation</td>
<td>Pseudo-Random Number Generators</td>
</tr>
<tr>
<td>Stream Ciphers</td>
<td></td>
</tr>
</tbody>
</table>

One option: Natural Randomness

Thermal noise from a semiconductor resistor.
Atmospheric noise.
Quantum-mechanical phenomena.
Tossing a coin.
One option: Natural Randomness

- Thermal noise from a semiconductor resistor.
One option: Natural Randomness

- Thermal noise from a semiconductor resistor.
- Atmospheric noise.
One option: Natural Randomness

- Thermal noise from a semiconductor resistor.
- Atmospheric noise.
- Quantum-mechanical phenomena.
One option: Natural Randomness

- Thermal noise from a semiconductor resistor.
- Atmospheric noise.
- Quantum-mechanical phenomena.
- Tossing a coin.
Why Natural Randomness is not useful?

- Difficulty of sampling.
- Difficulty of synchronizing when the sender and the receiver are far apart.
Why Natural Randomness is not useful?

- Difficulty of sampling.
Why Natural Randomness is not useful?

- Difficulty of sampling.
- Difficulty of synchronizing when the sender and the receiver are far apart.
Pragmatic Solution

A Finite State Machine.
A seed (called the secret key) characterizes the initial state.
The same seed generates the same output sequence.
The seed can be shared between the sender and the receiver.
A Finite State Machine.
Pragmatic Solution

- A Finite State Machine.
- A seed (called the secret key) characterizes the initial state.
A Finite State Machine.

A seed (called the secret key) characterizes the initial state.

Same seed generates the same output sequence.
Pragmatic Solution

- A Finite State Machine.
- A seed (called the secret key) characterizes the initial state.
- Same seed generates the same output sequence.
- Seed can be shared between the sender and the receiver.
Inherent Limitations

- Each state transition of the FSM gives one new output.
- FSM has finite no. of states.
- So the output sequence must have a period.
- One Time Pad cannot be realized in practice.

Goal: short seed, but long keystream.
Inherent Limitations

- Each state transition of the FSM gives one new output.
Inherent Limitations

- Each state transition of the FSM gives one new output.
- FSM has finite no. of states.
Inherent Limitations

- Each state transition of the FSM gives one new output.
- FSM has finite no. of states.
- So the output sequence must have a period.
Inherent Limitations

- Each state transition of the FSM gives one new output.
- FSM has finite no. of states.
- So the output sequence must have a period.
- One Time Pad cannot be realized in practice.
Inherent Limitations

- Each state transition of the FSM gives one new output.
- FSM has finite no. of states.
- So the output sequence must have a period.
- One Time Pad cannot be realized in practice.
- Goal: short seed, but long keystream.
Linear Congruential Generator

\[ x_n = ax_{n-1} + b \mod m. \]

- \( x_0 \) is the initial seed.
- \( a, b, m \) are parameters.
- Example: C library function `rand()`.
- Suitable for experimental purposes, but cryptographically not secure.
- Same is true for any polynomial congruential generator.
Randomness
Random Number Generation
Stream Ciphers

Natural Random Number Generators
Pseudo-Random Number Generators

Linear Congruential Generator

\[ x_n = ax_{n-1} + b \pmod{m}. \]
Linear Congruential Generator

\[ x_n = ax_{n-1} + b \pmod{m}. \]

- \( x_0 \) is the initial seed.
Linear Congruential Generator

\[ x_n = ax_{n-1} + b \pmod{m}. \]

- \( x_0 \) is the initial seed.
- \( a, b, m \) are parameters.
Randomness
Random Number Generation
Stream Ciphers

Natural Random Number Generators
Pseudo-Random Number Generators

Linear Congruential Generator

\[ x_n = ax_{n-1} + b \pmod{m}. \]

- \( x_0 \) is the initial seed.
- \( a, b, m \) are parameters.
- Example: C library function \textit{rand}().
Linear Congruential Generator

\[ x_n = ax_{n-1} + b \mod m. \]

- \( x_0 \) is the initial seed.
- \( a, b, m \) are parameters.
- Example: C library function \( \text{rand}() \).
- Suitable for experimental purposes, but cryptographically not secure.
Linear Congruential Generator

\[ x_n = ax_{n-1} + b \pmod{m}. \]

- \( x_0 \) is the initial seed.
- \( a, b, m \) are parameters.
- Example: C library function `rand()`.
- Suitable for experimental purposes, but cryptographically not secure.
- Same is true for any polynomial congruential generator.
Blum-Blum-Shub (BBS) Generator

Choose two large primes $p$ and $q$ both congruent to 3 mod 4. Set $n = pq$ and choose a random integer $x$ relatively prime to $n$. Set initial seed $x_0 = x^2 \pmod{n}$. $j$-th output is given by $x_j = x_{2j-1} \pmod{n}$. Has provable security, but too slow for practical use.
Choose two large primes $p, q$ both congruent to 3 mod 4.
Blum-Blum-Shub (BBS) Generator

- Choose two large primes \( p, q \) both congruent to 3 mod 4.
- Set \( n = pq \) and choose a random integer \( x \) relatively prime to \( n \).
Blum-Blum-Shub (BBS) Generator

- Choose two large primes $p, q$ both congruent to $3 \mod 4$.
- Set $n = pq$ and choose a random integer $x$ relatively prime to $n$.
- Set initial seed $x_0 = x^2 \pmod{n}$.
Choose two large primes $p, q$ both congruent to 3 mod 4.
Set $n = pq$ and choose a random integer $x$ relatively prime to $n$.
Set initial seed $x_0 = x^2 \pmod{n}$.
$j$-th output is given by $x_j = x_{j-1}^2 \pmod{n}$.
Blum-Blum-Shub (BBS) Generator

- Choose two large primes \( p, q \) both congruent to 3 mod 4.
- Set \( n = pq \) and choose a random integer \( x \) relatively prime to \( n \).
- Set initial seed \( x_0 = x^2 \pmod{n} \).
- \( j \)-th output is given by \( x_j = x_{j-1}^2 \pmod{n} \).
- Has provable security, but too slow for practical use.
Roadmap

1 Randomness
   • Defining Randomness
   • Testing Randomness
   • Cryptographic Randomness

2 Random Number Generation
   • Natural Random Number Generators
   • Pseudo-Random Number Generators

3 Stream Ciphers
   • Hardware Stream Ciphers
   • Software Stream Ciphers
   • Distinguisher
General Model of Stream Ciphers
The same key always produces the same keystream. Repeated use of the same key is just as bad as reusing a one-time pad. As a remedy, the IV is combined with the secret key to form the effective key for the corresponding session of the cipher, called a session key. Different session keys make the output of the stream cipher different in each session, even if the same key is used.
Need for Initialization Vector (IV)

- The same key always produces the same keystream.
Need for Initialization Vector (IV)

- The same key always produces the same keystream.
- Repeated use of the same key is just as bad as reusing a one-time pad.
The same key always produces the same keystream.

Repeated use of the same key is just as bad as reusing a one-time pad.

As a remedy, the IV is combined with the secret key to form the effective key for the corresponding session of the cipher, called a session key.
The same key always produces the same keystream.

Repeated use of the same key is just as bad as reusing a one-time pad.

As a remedy, the IV is combined with the secret key to form the effective key for the corresponding session of the cipher, called a session key.

Different session keys make the output of the stream cipher different in each session, even if the same key is used.
Hardware vs. Software Stream Ciphers

Hardware Stream Ciphers:
- LFSRs are used as linear elements.
- Combining functions (may be with some amount of memory) are used as nonlinear elements.

Software Stream Ciphers:
- May use word-based LFSR / NFSRs.
- May use arrays, modular additions and other operators.
Hardware vs. Software Stream Ciphers

- Hardware Stream Ciphers.

- Combining functions (may be with some amount of memory) are used as nonlinear elements.

- Software Stream Ciphers.
  - May use word-based LFSR / NFSRs.
  - May use arrays, modular additions and other operators.
Hardware vs. Software Stream Ciphers

- Hardware Stream Ciphers.
  - LFSRs are used as linear elements.
Hardware vs. Software Stream Ciphers

- Hardware Stream Ciphers.
  - LFSRs are used as linear elements.
  - Combining functions (may be with some amount of memory) are used as nonlinear elements.
Hardware vs. Software Stream Ciphers

- Hardware Stream Ciphers.
  - LFSRs are used as linear elements.
  - Combining functions (may be with some amount of memory) are used as nonlinear elements.

- Software Stream Ciphers.
Hardware vs. Software Stream Ciphers

- **Hardware Stream Ciphers.**
  - LFSRs are used as linear elements.
  - Combining functions (may be with some amount of memory) are used as nonlinear elements.

- **Software Stream Ciphers.**
  - May use word-based LFSR / NFSRs.
Hardware vs. Software Stream Ciphers

- **Hardware Stream Ciphers.**
  - LFSRs are used as linear elements.
  - Combining functions (may be with some amount of memory) are used as nonlinear elements.

- **Software Stream Ciphers.**
  - May use word-based LFSR / NFSRs.
  - May use arrays, modular additions and other operators.
Bit-oriented LFSR

Recurrence Relation:

\[ x_{n+6} = x_{n+4} \oplus x_{n+1} \oplus x_n \]

Polynomial over \( \mathbb{F}_2 \):

\[ x^6 + x^4 + x^1 + 1 \]

Figure: LFSR: one step evolution
Bit-oriented LFSR

Recurrence Relation: \( x_{n+6} = x_{n+4} \oplus x_{n+1} \oplus x_n \)

Figure: LFSR: one step evolution
Bit-oriented LFSR

![LFSR Diagram]

**Figure:** LFSR: one step evolution

- Recurrence Relation: $x_{n+6} = x_{n+4} \oplus x_{n+1} \oplus x_n$
- Polynomial over $GF(2)$: $x^6 + x^4 + x^1 + 1$
Bit-oriented LFSR (cont’d.)

Primitive polynomial provides maximum length cycle, $2^d - 1$ for degree $d$. Well known as $m$-sequence. By itself, not cryptographically secure, but useful building block for pseudo-randomness. Easy and efficient implementation in hardware, using registers (Flip-Flops) and simple logic gates. Deep mathematical development for a long time.
Bit-oriented LFSR (cont’d.)

- Primitive polynomial provides maximum length cycle, $2^d - 1$ for degree $d$. Well known as $m$-sequence.
**Bit-oriented LFSR (cont’d.)**

- Primitive polynomial provides maximum length cycle, $2^d - 1$ for degree $d$. Well known as $m$-sequence.
- By itself, not cryptographically secure, but useful building block for pseudo-randomness.
Primitive polynomial provides maximum length cycle, $2^d - 1$ for degree $d$. Well known as $m$-sequence.
By itself, not cryptographically secure, but useful building block for pseudo-randomness.
Easy and efficient implementation in hardware, using registers (Flip-Flops) and simple logic gates.
Bit-oriented LFSR (cont’d.)

- Primitive polynomial provides maximum length cycle, $2^d - 1$ for degree $d$. Well known as $m$-sequence.
- By itself, not cryptographically secure, but useful building block for pseudo-randomness.
- Easy and efficient implementation in hardware, using registers (Flip-Flops) and simple logic gates.
- Deep mathematical development for a long time.
Attacking the LFSR-based PRNGs

Suppose we know the segment 011010111100 of a keystream sequence. We also know that it is generated by some LFSR. We do not necessarily know the length of the recurrence. We need to determine the coefficients.
Suppose we know the segment 011010111100 of a keystream sequence.
Suppose we know the segment 011010111100 of a keystream sequence.

We also know that it is generated by some LFSR.
Attacking the LFSR-based PRNGs

- Suppose we know the segment 011010111100 of a keystream sequence.
- We also know that it is generated by some LFSR.
- We do not necessarily know the length of the recurrence.
Suppose we know the segment 011010111100 of a keystream sequence.

We also know that it is generated by some LFSR.

We do not necessarily know the length of the recurrence.

We need to determine the coefficients.
Try with Length 2

\[ x_{n+2} = c_0 x_n + c_1 x_{n+1}. \]
Try with Length 2

\[ x_{n+2} = c_0 x_n + c_1 x_{n+1}. \]

\[
\begin{bmatrix}
0 & 1 \\
1 & 1
\end{bmatrix}
\begin{bmatrix}
c_0 \\
c_1
\end{bmatrix} =
\begin{bmatrix}
1 \\
0
\end{bmatrix}
\]

Solution:
\[ c_0 = 1, \quad c_1 = 1. \]

But \( x_6 \neq x_4 + x_5 \).
Try with Length 2

\[ x_{n+2} = c_0 x_n + c_1 x_{n+1}. \]

\[
\begin{bmatrix}
0 & 1 \\
1 & 1 
\end{bmatrix}
\begin{bmatrix}
c_0 \\
c_1 
\end{bmatrix}
= \begin{bmatrix} 1 \\ 0 \end{bmatrix}
\]

Solution: \( c_0 = 1, \ c_1 = 1. \)
Try with Length 2

\[ x_{n+2} = c_0 x_n + c_1 x_{n+1}. \]

\[
\begin{bmatrix}
0 & 1 \\
1 & 1
\end{bmatrix}
\begin{bmatrix}
c_0 \\
c_1
\end{bmatrix} = 
\begin{bmatrix}
1 \\
0
\end{bmatrix}
\]

Solution: \( c_0 = 1, \ c_1 = 1. \)
But \( x_6 \neq x_4 + x_5. \)
Try with Length 3

\[ x_{n+3} = c_0 x_n + c_1 x_{n+1} + c_2 x_{n+2}. \]
Try with Length 3

\[ x_{n+3} = c_0 x_n + c_1 x_{n+1} + c_2 x_{n+2}. \]

\[
\begin{bmatrix}
0 & 1 & 1 \\
1 & 1 & 0 \\
1 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
c_0 \\
c_1 \\
c_2 \\
\end{bmatrix}
= 
\begin{bmatrix}
0 \\
1 \\
0 \\
\end{bmatrix}
\]
Try with Length 3

\[ x_{n+3} = c_0 x_n + c_1 x_{n+1} + c_2 x_{n+2}. \]

\[
\begin{bmatrix}
0 & 1 & 1 \\
1 & 1 & 0 \\
1 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
c_0 \\
c_1 \\
c_2 \\
\end{bmatrix}
= 
\begin{bmatrix}
0 \\
1 \\
0 \\
\end{bmatrix}
\]

Solution: ?
Try with Length 4

\[ x_{n+4} = c_0 x_n + c_1 x_{n+1} + c_2 x_{n+2} + c_3 x_{n+3} . \]
Try with Length 4

\[ x_{n+4} = c_0 x_n + c_1 x_{n+1} + c_2 x_{n+2} + c_3 x_{n+3}. \]

\[
\begin{bmatrix}
0 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
c_0 \\
c_1 \\
c_2 \\
c_3
\end{bmatrix}
= 
\begin{bmatrix}
1 \\
0 \\
1 \\
1
\end{bmatrix}
\]

Solution:
\[ c_0 = 1, \quad c_1 = 1, \quad c_2 = 0, \quad c_3 = 0. \]
Try with Length 4

\[ X_{n+4} = c_0 X_n + c_1 X_{n+1} + c_2 X_{n+2} + c_3 X_{n+3}. \]

\[
\begin{bmatrix}
0 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
c_0 \\
c_1 \\
c_2 \\
c_3 \\
\end{bmatrix} =
\begin{bmatrix}
1 \\
0 \\
1 \\
1 \\
\end{bmatrix}
\]

Solution: \( c_0 = 1, c_1 = 1, c_2 = 0, c_3 = 0. \)
General Problem

\[ x_{n+m} = c_0 x_n + c_1 x_{n+1} + \ldots + c_{m-1} x_{n+m-1} \]
General Problem

\[ x_{n+m} = c_0 x_n + c_1 x_{n+1} + \ldots + c_{m-1} x_{n+m-1} \]

\[
\begin{bmatrix}
  x_1 & x_2 & \ldots & x_m \\
  x_2 & x_3 & \ldots & x_{m+1} \\
  \vdots & \vdots & \ddots & \vdots \\
  x_m & x_{m+1} & \ldots & x_{2m-1}
\end{bmatrix}
\begin{bmatrix}
  c_0 \\
  c_1 \\
  \vdots \\
  c_{m-1}
\end{bmatrix}
= 
\begin{bmatrix}
  x_{m+1} \\
  x_{m+2} \\
  \vdots \\
  x_{2m}
\end{bmatrix}
\]
Randomness
Random Number Generation
Stream Ciphers

General Problem

\[ x_{n+m} = c_0 x_n + c_1 x_{n+1} + \ldots + c_{m-1} x_{n+m-1} \]

\[
\begin{bmatrix}
  x_1 & x_2 & \ldots & x_m \\
  x_2 & x_3 & \ldots & x_{m+1} \\
  \vdots & \vdots & \ddots & \vdots \\
  x_m & x_{m+1} & \ldots & x_{2m-1}
\end{bmatrix}
\begin{bmatrix}
  c_0 \\
  c_1 \\
  \vdots \\
  c_{m-1}
\end{bmatrix}
= 
\begin{bmatrix}
  x_{m+1} \\
  x_{m+2} \\
  \vdots \\
  x_{2m}
\end{bmatrix}
\]

Result: The \( m \times m \) matrix is invertible mod2, iff there is no linear recurrence relation of length less than \( m \) that is satisfied by the \( 2m \) values \( x_1, x_2, \ldots, x_{2m} \).
Nonlinear Combiner Model

Take $n$ LFSRs of different length (may be pairwise prime). Initialize them with seeds. In each clock, take the $n$-many outputs from the LFSRs, which are fed as $n$-inputs to an $n$-variable Boolean function. May be some memory element is added.
Nonlinear Combiner Model

- Take $n$ LFSRs of different length (may be pairwise prime).
Nonlinear Combiner Model

- Take $n$ LFSRs of different length (may be pairwise prime).
- Initialize them with seeds.
Nonlinear Combiner Model

- Take $n$ LFSRs of different length (may be pairwise prime).
- Initialize them with seeds.
- In each clock, take the $n$-many outputs from the LFSRs, which are fed as $n$-inputs to an $n$-variable Boolean function.
Nonlinear Combiner Model

- Take $n$ LFSRs of different length (may be pairwise prime).
- Initialize them with seeds.
- In each clock, take the $n$-many outputs from the LFSRs, which are fed as $n$-inputs to an $n$-variable Boolean function.
- May be some memory element is added.
Nonlinear Filter-Generator Model

Take one LFSR. Initialize that with a seed. In each clock, take the $n$-many outputs from the LFSR from different locations, which are fed as $n$-inputs to an $n$-variable Boolean function. May be considered with additional memory element. The Boolean function and memory together form a Finite State Machine.
Nonlinear Filter-Generator Model

- Take one LFSR.
Nonlinear Filter-Generator Model

- Take one LFSR.
- Initialize that with a seed.
Nonlinear Filter-Generator Model

- Take one LFSR.
- Initialize that with a seed.
- In each clock, take the $n$-many outputs from the LFSR from different locations, which are fed as $n$-inputs to an $n$-variable Boolean function.
Nonlinear Filter-Generator Model

- Take one LFSR.
- Initialize that with a seed.
- In each clock, take the $n$-many outputs from the LFSR from different locations, which are fed as $n$-inputs to an $n$-variable Boolean function.
- May be considered with additional memory element.
Nonlinear Filter-Generator Model

- Take one LFSR.
- Initialize that with a seed.
- In each clock, take the $n$-many outputs from the LFSR from different locations, which are fed as $n$-inputs to an $n$-variable Boolean function.
- May be considered with additional memory element.
- The Boolean function and memory together form a Finite State Machine.
<table>
<thead>
<tr>
<th>Boolean Function: Cryptographic Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Randomness</strong></td>
</tr>
<tr>
<td>Random Number Generation</td>
</tr>
<tr>
<td>Stream Ciphers</td>
</tr>
<tr>
<td><strong>Hardware Stream Ciphers</strong></td>
</tr>
<tr>
<td>Software Stream Ciphers</td>
</tr>
<tr>
<td>Distinguisher</td>
</tr>
</tbody>
</table>

### Cryptographic Properties

- **Balancedness**: Necessary to achieve a pseudo-random sequence.
- **Algebraic Degree**: To achieve high linear complexity.
- **Nonlinearity**: For higher confusion and resistance against Best Affine Approximation (BAA) Attack and linear cryptanalysis.
- **Autocorrelation**: To achieve higher diffusion, and to resist differential cryptanalysis.
- **Crosscorrelation Immunity**: To resist correlation attack.
- **Algebraic Immunity**: To resist algebraic attack.
Boolean Function: Cryptographic Properties

- **BALANCEDNESS**: Necessary to achieve Pseudo-Random sequence
**Boolean Function: Cryptographic Properties**

- **BALANCEDNESS**: Necessary to achieve Pseudo-Random sequence
- **ALGEBRAIC DEGREE**: To achieve high Linear Complexity
**Boolean Function: Cryptographic Properties**

- **BALANCEDNESS**: Necessary to achieve Pseudo-Random sequence
- **ALGEBRAIC DEGREE**: To achieve high Linear Complexity
- **NONLINEARITY**: For higher Confusion and resistance against: Best Affine Approximation (BAA) Attack and Linear Cryptanalysis.
Boolean Function: Cryptographic Properties

- **BALANCEDNESS**: Necessary to achieve Pseudo-Random sequence
- **ALGEBRAIC DEGREE**: To achieve high Linear Complexity
- **NONLINEARITY**: For higher Confusion and resistance against: Best Affine Approximation (BAA) Attack and Linear Cryptanalysis.
- **AUTOCORRELATION**: To achieve higher Diffusion, and to resist Differential Cryptanalysis.
Boolean Function: Cryptographic Properties

- **BALANCEDNESS**: Necessary to achieve Pseudo-Random sequence
- **ALGEBRAIC DEGREE**: To achieve high Linear Complexity
- **NONLINEARITY**: For higher Confusion and resistance against: Best Affine Approximation (BAA) Attack and Linear Cryptanalysis.
- **AUTOCORRELATION**: To achieve higher Diffusion, and to resist Differential Cryptanalysis.
- **CORRELATION IMMUNITY**: To resist Correlation Attack
Boolean Function: Cryptographic Properties

- **Balancedness**: Necessary to achieve Pseudo-Random sequence
- **Algebraic Degree**: To achieve high Linear Complexity
- **Nonlinearity**: For higher Confusion and resistance against: Best Affine Approximation (BAA) Attack and Linear Cryptanalysis.
- **Autocorrelation**: To achieve higher Diffusion, and to resist Differential Cryptanalysis.
- **Correlation Immunity**: To resist Correlation Attack
- **Algebraic Immunity**: To resist Algebraic Attack
Hardware Stream Ciphers: Current Trends

- Nonlinear Filter Generator Model With Memory
- More than one bit processed together (32-bit words)
- Use LFSRs over larger fields: need the LFSR evolution operations to be efficient.
- $\text{GF}(2^{32})$ or $\text{GF}(2^{31}-1)$ to relate with 32-bit words of modern processors. Are we moving towards 64-bit words?
- FSM contains S-boxes and Registers
- Registers are memory words
- S-boxes are multiple output Boolean functions
Nonlinear Filter Generator Model With Memory.
Nonlinear Filter Generator Model With Memory.
More than one bit processed together (32-bit words)
Hardware Stream Ciphers: Current Trends

- Nonlinear Filter Generator Model With Memory.
- More than one bit processed together (32-bit words)
- Use LFSRs over larger fields: need the LFSR evolution operations to be efficient.
Nonlinear Filter Generator Model With Memory.
More than one bit processed together (32-bit words)
Use LFSRs over larger fields: need the LFSR evolution operations to be efficient.
$GF(2^{32})$ or $GF(2^{31} - 1)$ to relate with 32-bit words of modern processors. Are we moving towards 64-bit words?
Nonlinear Filter Generator Model With Memory.
More than one bit processed together (32-bit words)
Use LFSRs over larger fields: need the LFSR evolution operations to be efficient.
$GF(2^{32})$ or $GF(2^{31} – 1)$ to relate with 32-bit words of modern processors. Are we moving towards 64-bit words?
FSM contains S-boxes and Registers.
Hardware Stream Ciphers: Current Trends

- Nonlinear Filter Generator Model With Memory.
- More than one bit processed together (32-bit words)
- Use LFSRs over larger fields: need the LFSR evolution operations to be efficient.
  \( GF(2^{32}) \) or \( GF(2^{31} - 1) \) to relate with 32-bit words of modern processors. Are we moving towards 64-bit words?
- FSM contains S-boxes and Registers.
- Registers are memory words.
Randomness
Random Number Generation
Stream Ciphers

Hardware Stream Ciphers: Current Trends

- Nonlinear Filter Generator Model With Memory.
- More than one bit processed together (32-bit words)
- Use LFSRs over larger fields: need the LFSR evolution operations to be efficient.
- $GF(2^{32})$ or $GF(2^{31} - 1)$ to relate with 32-bit words of modern processors. Are we moving towards 64-bit words?
- FSM contains S-boxes and Registers.
- Registers are memory words.
- S-boxes are multiple output Boolean functions.
Initially, stream ciphers were targeted towards hardware only. Later, software stream ciphers became popular due to their speed and efficiency compared to software implementation of block ciphers. Typically consists of two modules:

- KSA: key $\times$ IV $\rightarrow$ internal state
- PRGA: internal state $\rightarrow$ keystream word.
Initially, stream ciphers were targeted towards hardware only.
Initially, stream ciphers were targeted towards hardware only.

Later, software stream ciphers became popular due to their speed and efficiency compared to software implementation of block ciphers.
Initially, stream ciphers were targeted towards hardware only.

Later, software stream ciphers became popular due to their speed and efficiency compared to software implementation of block ciphers.

Typically consists of two modules:

- **KSA**: key $\times$ IV $\rightarrow$ internal state and
- **PRGA**: internal state $\rightarrow$ keystream word.
### An Example: RC4 (Ron Rivest, 1987)

- **Wide commercial applications**: SSL, TLS, WEP, WPA, AOCE, Microsoft Windows, Lotus Notes, Oracle Secure SQL etc.
- Generally used with 5 to 16 bytes key, though provision for 256 bytes key is there.
- Uses a permutation over $\mathbb{Z}_{256}$ as the internal state.
- Operations: Swaps and Modulo 256 additions.
An Example: RC4 (Ron Rivest, 1987)

- Wide commercial applications SSL, TLS, WEP, WPA, AOCE, Microsoft Windows, Lotus Notes, Oracle Secure SQL etc.
An Example: RC4 (Ron Rivest, 1987)

- Wide commercial applications SSL, TLS, WEP, WPA, AOCE, Microsoft Windows, Lotus Notes, Oracle Secure SQL etc.
- Generally used with 5 to 16 bytes key, though provision for 256 bytes key is there.
An Example: RC4 (Ron Rivest, 1987)

- Wide commercial applications SSL, TLS, WEP, WPA, AOCE, Microsoft Windows, Lotus Notes, Oracle Secure SQL etc.
- Generally used with 5 to 16 bytes key, though provision for 256 bytes key is there.
- Uses a permutation over $\mathbb{Z}_{256}$ as the internal state.
Wide commercial applications SSL, TLS, WEP, WPA, AOCE, Microsoft Windows, Lotus Notes, Oracle Secure SQL etc.

Generally used with 5 to 16 bytes key, though provision for 256 bytes key is there.

Uses a permutation over $\mathbb{Z}_{256}$ as the internal state.

Operations: Swaps and Modulo 256 additions.
### RC4 KSA

#### Initialize S-box to identity permutation of \( \{0, 1, \ldots, 255\} \)

#### Initialize counter: \( j = 0 \);

#### for \( i = 0, \ldots, 255 \)

- \( j = j + S[i] + K[i] \);
- Swap: \( S[i] \leftrightarrow S[j] \);


- Initialize the counters: $i = j = 0$;
- While you need keystream bytes
  - Increment counters $i = i + 1$ and $j = j + S[i]$;
  - Swap $S[i] \leftrightarrow S[j]$;
  - Output $Z = S[S[i] + S[j]]$;
Software Stream Ciphers: Current Trends

Word oriented design.
Complicated Functions and Operations.
Huge Internal State.
Software Stream Ciphers: Current Trends

- Word oriented design.
Software Stream Ciphers: Current Trends

- Word oriented design.
- Complicated Functions and Operations.
Software Stream Ciphers: Current Trends

- Word oriented design.
- Complicated Functions and Operations.
- Huge Internal State.
Basic Idea

An event that distinguishes the keystream from a uniformly random stream. For a stream cipher, the event is based on some combination of the keystream bits. The attack complexity is given by the number of samples required for a given success probability.
Basic Idea

- An event that distinguishes the keystream from a uniformly random stream.
Basic Idea

- An event that distinguishes the keystream from a uniformly random stream.
- For a stream cipher, the event is based on some combination of the keystream bits.
Basic Idea

- An event that distinguishes the keystream from a uniformly random stream.
- For a stream cipher, the event is based on some combination of the keystream bits.
- The attack complexity is given by the number of samples required for a given success probability.
The Setup

Define $X_r = 1$, if $A$ occurs in $r$-th sample, else it is 0.

If we observe $n$ samples, $\sum_{r=1}^{n} X_r \sim \text{Binomial}(n, p)$.

When $X_r$'s are i.i.d. and $n$ is large enough, $\sum_{r=1}^{n} X_r \sim \text{Normal}(np, np(1-p))$. 
The Setup

Event $A$, $P(A) = p$. 

Define $X_r = 1$, if $A$ occurs in the $r$-th sample, else it is 0.

If we observe $n$ samples, $n \sum X_r \sim B(n, p)$.

When $X_r$'s are i.i.d. and $n$ is large enough, $n \sum X_r \sim N(np, np(1-p))$. 
The Setup

Event $A$, $P(A) = p$.

Define $X_r = 1$, if $A$ occurs in $r$-th sample, else it is $0$. 
The Setup

Event $A$, $P(A) = p$.

Define $X_r = 1$, if $A$ occurs in $r$-th sample, else it is 0.

If we observe $n$ samples,
The Setup

Event $A$, $P(A) = p$.

Define $X_r = 1$, if $A$ occurs in $r$-th sample, else it is 0.

If we observe $n$ samples,

$$\sum_{r=1}^{n} X_r \sim \mathcal{B}(n, p).$$
The Setup

Event $A$, $P(A) = p$.

Define $X_r = 1$, if $A$ occurs in $r$-th sample, else it is 0.

If we observe $n$ samples,

$$
\sum_{r=1}^{n} X_r \sim \text{B}(n, p).
$$

When $X_r$’s are i.i.d. and $n$ is large enough,
The Setup

Event $A$, $P(A) = p$.

Define $X_r = 1$, if $A$ occurs in $r$-th sample, else it is 0.

If we observe $n$ samples,

$$\sum_{r=1}^{n} X_r \sim \mathcal{B}(n, p).$$

When $X_r$’s are i.i.d. and $n$ is large enough,

$$\sum_{r=1}^{n} X_r \sim \mathcal{N}(np, np(1 - p)).$$
Hypothesis Testing Approach

\[ H_0 : p = p_0 (1 + \epsilon), \epsilon > 0 \]
against

\[ H_1 : p = p_0. \]
Hypothesis Testing Approach

Test

\[ H_0 : p = p_0(1 + \epsilon), \epsilon > 0, \]
Test

\[ H_0 : p = p_0(1 + \epsilon), \epsilon > 0, \]

against

\[ H_1 : p = p_0. \]
Bounding the Errors

The objective is to find a threshold $c$ in $[np_0, np_0(1 + \epsilon)]$ such that

$$P(n \sum_{r=1}^{n} X_r \leq c | H_0) \leq \alpha$$

and

$$P(n \sum_{r=1}^{n} X_r > c | H_1) \leq \beta.$$
The objective is to find a threshold $c$ in $[np_0, np_0(1 + \epsilon)]$ such that
Bounding the Errors

The objective is to find a threshold $c$ in $[np_0, np_0(1 + \epsilon)]$ such that

$$P\left(\sum_{r=1}^{n} X_r \leq c \mid H_0\right) \leq \alpha$$
Bounding the Errors

The objective is to find a threshold $c$ in $[np_0, np_0(1 + \epsilon)]$ such that

$$P \left( \sum_{r=1}^{n} X_r \leq c \mid H_0 \right) \leq \alpha$$

and

$$P \left( \sum_{r=1}^{n} X_r > c \mid H_1 \right) \leq \beta.$$
Necessary Condition

For such a $c$ to exist,
Necessary Condition

For such a $c$ to exist,

$$np_0(1 + \epsilon) - \kappa_1 \sigma_1 > np_0 + \kappa_2 \sigma_2,$$
Necessary Condition

For such a \( c \) to exist,

\[
np_0(1 + \epsilon) - \kappa_1 \sigma_1 > np_0 + \kappa_2 \sigma_2,
\]

where

\[
\sigma_1^2 = np_0(1 + \epsilon)(1 - p_0(1 + \epsilon)),
\]
\[
\sigma_2^2 = np_0(1 - p_0),
\]
\[
\Phi(-\kappa_1) = \alpha
\]
and \( \Phi(\kappa_2) = 1 - \beta \).
When $p_0, \epsilon \ll 1$, 

$$n > \frac{(\kappa_1 + \kappa_2)^2}{p_0 \epsilon^2}.$$
How Many Samples Required?

When $p_0, \epsilon \ll 1$,

$$n > \frac{(\kappa_1 + \kappa_2)^2}{p_0 \epsilon^2}.$$  

$\kappa_1 = \kappa_2 = 0.5$ gives $\alpha = \beta = 1 - 0.6915$ and at least $\frac{1}{p_0 \epsilon^2}$ samples are required.
<table>
<thead>
<tr>
<th>Randomness</th>
<th>Hardware Stream Ciphers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Number Generation</td>
<td>Software Stream Ciphers</td>
</tr>
<tr>
<td>Stream Ciphers</td>
<td>Distinguisher</td>
</tr>
</tbody>
</table>

**Example of a Distinguisher**

- RC4 2nd byte.
- Attack on Broadcast.
Example of a Distinguisher

- RC4 2nd byte.
Example of a Distinguisher

- RC4 2nd byte.
- Attack on Broadcast.
I end my talk here ...

Thank You

Homepage: http://www.goutampaul.com
Email: goutam.paul@ieee.org